Automated Program Verification and Testing 15414/15614 Fall 2016 Lecture 3: Practical SAT Solving

Matt Fredrikson
mfredrik@cs.cmu.edu

October 17, 2016

Review: Propositional Semantics

Goal: Give meaning to propositional formulas

Assign Boolean truth values to (formula, interpretation) pairs

Formula F + **Interpretation** I = **Truth Value** (*true*, *false*)

Note: we often abbreviate *true* by 1 and *false* by 0

Interpretation

An interpretation I for propositional formula F maps every propositional variable appearing in F to a truth value, i.e.:

$$I = \{P \mapsto \textit{true}, Q \mapsto \textit{false}, R \mapsto \textit{false}, \ldots\}$$

Review: Interpretations

Satisfying Interpretation

I is a *satisfying interpretation* of a propositional formula F if F is *true* under I. We denote this with the notation:

$$I \models F$$

Falsifying Interpretation

I is a *falsifying interpretation* of a propositional formula F if F is *false* under I. We denote this with the notation:

$$I \not\models F$$

Review: Conjunctive Normal Form (CNF)

Take the form:

$$\bigwedge_i \bigvee_j P_{ij}$$

To convert to CNF:

1. Convert to NNF
2. Distribute $\vee$ over $\wedge$

Naive approach has exponential blowup

Tseitin's transformation: linear increase in formula size

$$\begin{aligned}
\langle \textit{atom} \rangle &::= \top \mid \bot \mid P, Q, \ldots \\
\langle \textit{literal} \rangle &::= \langle \textit{atom} \rangle \mid \neg \langle \textit{atom} \rangle \\
\langle \textit{clause} \rangle &::= \langle \textit{literal} \rangle \\
&\quad\; \mid \langle \textit{literal} \rangle \vee \langle \textit{clause} \rangle \\
\langle \textit{formula} \rangle &::= \langle \textit{clause} \rangle \\
&\quad\; \mid \langle \textit{clause} \rangle \wedge \langle \textit{formula} \rangle
\end{aligned}$$

Satisfiability Problem

SAT Problem

Given a propositional formula F, decide whether there exists an interpretation I such that $I \models F$.

3SAT was the first established NP-Complete problem (Cook, 1971)

Most important logical problems can be reduced to SAT

- Validity
- Entailment
- Equivalence

CNF Notation

All of the algorithms we talk about assume that formulas are in CNF

We'll refer to a formula as a set of clauses $F = \{C_1, \ldots, C_n\}$

Likewise, clauses as sets of literals

$$(P \vee Q) \wedge (Q \rightarrow \neg P) \quad \{\{P, Q\}, \{\neg Q, \neg P\}\}$$

Some convenient notation:

- $C_i\{P \mapsto F\}$: C_i with F substituted for P
- $C_i[P]$: P appears positive in C_i, i.e., $C_i = \{\ldots, P, \ldots\}$
- $C_i[\neg P]$: P appears negated in C_i, i.e., $C_i = \{\ldots, \neg P, \ldots\}$
- $C_i \vee C_j$: union of C_i and C_j, $C_i \cup C_j$
- $F_i \wedge F_j$: union of F_i and F_j, $F_i \cup F_j$

Resolution

Single inference rule:

$$\frac{C_1[P] \qquad C_2[\neg P]}{C_1\{P \mapsto \bot\} \vee C_2\{\neg P \mapsto \bot\}}$$

Given two clauses that share variable P but disagree on its value:

1. If P is *true*, then some other literal in C_2 must be true
2. If P is *false*, then some other literal in C_1 must be true
3. Therefore, *resolve* on P in both clauses by removing it
4. $C_1\{P \mapsto \bot\} \vee C_2\{\neg P \mapsto \bot\}$ is called the *resolvent*

If $C_1\{P \mapsto \bot\} \vee C_2\{\neg P \mapsto \bot\} = \bot \vee \bot = \bot$:

1. Then $C_1 \wedge C_2$ is unsatisfiable
2. Any CNF containing $\{C_1, C_2\}$ is unsatisfiable

Resolution Procedure

function Resolution(F)
  F' = $\varnothing$
  repeat
    $F \leftarrow F \cup F'$
    for all $C_i, C_j \in F$ **do**
      C' = Resolve(C_i, C_j)
      if C' = $\bot$ **then**
        return *unsat*
      end if
      $F' \leftarrow F' \cup \{C'\}$
    end for
  until $F' \subseteq F$
  return *sat*
end function

1. For each round, compute all possible resolvents
2. F' holds set of all resolvents
3. At each round, update F to contain past resolvents
4. Repeat resolution on updated F
5. Terminate when:
   - Encounter $\bot$ resolvent
   - Don't find anything new to add to F

Resolution: Example

$$\boxed{(P \vee Q) \;\wedge\; (P \rightarrow R) \;\wedge\; (Q \rightarrow R) \;\wedge\; \neg R}$$

$$\underbrace{(P \vee Q)}_{C_1} \wedge \underbrace{(\neg P \vee R)}_{C_2} \wedge \underbrace{(\neg Q \vee R)}_{C_3} \wedge \underbrace{\neg R}_{C_4}$$

1	$P \vee Q$	
2	$\neg P \vee R$	
3	$\neg Q \vee R$	
4	$\neg R$	
5	$Q \vee R$	1 & 2
7	$\neg P$	2 & 4
8	$\neg Q$	3 & 4

9	R	3 &5
10	Q	4 &5
11	P	1 &8
12	$\bot$	4 &9

Resolution: Properties

Why is resolution particularly bad for large problems?

Hint: What does this technique build along the way?

Space complexity: exp$(O(N))$

Example: m pigeons won't go into n holes when $m > n$

- $p_{i,j}$: pigeon i goes in hole j
- $p_{i,1} \vee p_{i,2} \vee \cdots \vee p_{i,n}$: every pigeon i gets a hole
- $\neg p_{i,j} \vee \neg p_{i',j}$: no hole j gets two pigeons $i \neq i'$
- Resolution proof size: exp$(\Omega(N))$

Partial Interpretations

Starting from an empty interpretation:

- Extend for each variable
- No direct modifications to literals in formula

More flexibility in implementation strategy (more on this later)

If I is a *partial* interpretation, literals ℓ can be *true*, *false*, *undef*:

- *true* (satisfied): $I \models \ell$
- *false* (conflicting): $I \not\models \ell$
- *undef*: var$(\ell) \notin I$

Given a clause C and interpretation I:

- C is *true* under I iff $I \models C$
- C is *false* under I iff $I \not\models C$
- C is *unit* under I iff $C = C' \vee \ell$, $I \not\models C$, ℓ is *undef*
- Otherwise it is *undef*

Example

$$\boxed{I = \{P_1 \mapsto 1, P_2 \mapsto 0, P_4 \mapsto 1\}}$$

$P_1 \vee P_3 \vee \neg P_4$	*satisfied*
$\neg P_1 \vee P_2$	*conflicting*
$\neg P_1 \vee \neg P_4 \vee P_3$	*unit*
$\neg P_1 \vee P_3 \vee P_5$	*undef*

Decision Procedure as a Transition System

Transition system is a binary relation over **states**

Transitions are induced by *guarded* transition rules

Procedure State

The possible states are:

- *sat*
- *unsat*
- $[I] \parallel F$

Where $[I]$ is an *ordered* interpretation, F is a CNF.

Initial state: $[\varnothing] \parallel F$

Final states: *sat*, *unsat*

Ex. intermediate states:

- $[\varnothing] \parallel F_1, C$: empty interpretation, $F = F_1 \wedge C$
- $[I_1, \overline{P}, I_2] \parallel F$: interp. assigns I_1 first, then $P \mapsto 0$, then I_2

Basic Search

Decision Rule

$$[I] \parallel F \hookrightarrow [I, P^\circ] \parallel F \text{ if } \begin{cases} P \text{ occurs in } F \\ P \text{ unassigned in } I \end{cases}$$

Backtrack Rule

$$[I_1, P^\circ, I_2] \parallel F \hookrightarrow [I_1, \overline{P}] \parallel F \text{ if } \begin{cases} [I_1, P, I_2] \not\models F \\ P \text{ last decision in interp.} \end{cases}$$

Sat Rule

$$[I] \parallel F \hookrightarrow \textit{sat} \text{ if } [I] \models F$$

Unsat Rule

$$[I] \parallel F \hookrightarrow \textit{unsat} \text{ if } \begin{cases} [I] \not\models F \\ \text{No decisions in } I \end{cases}$$

Example

$$F := \begin{array}{lll} C_1 = \neg P_1 \lor P_2 & C_2 = \neg P_3 \lor P_4 & C_3 = \neg P_6 \lor \neg P_5 \lor \neg P_2 \\ C_4 = \neg P_5 \lor P_6 & C_5 = P_5 \lor P_7 & C_6 = \neg P_1 \lor P_5 \lor P_7 \end{array}$$

I	Rule
P_2°	Decide
P_2°, P_4°	Decide
$P_2^\circ, P_4^\circ, P_5^\circ$	Decide
$P_2^\circ, P_4^\circ, P_5^\circ, P_6^\circ$	Decide
$P_2^\circ, P_4^\circ, P_5^\circ, \overline{P_6}$	Backtrack
$P_2^\circ, P_4^\circ, \overline{P_5}$	Backtrack
$P_2^\circ, P_4^\circ, \overline{P_5}, P_7^\circ$	Decide
$P_2^\circ, P_4^\circ, \overline{P_5}, P_7^\circ$	Sat

Unit Propagation

Recall *unit* clauses. For an interpretation I and clause C,

- I does not satisfy C
- All but one literals in C are assigned

I implies an assignment for the unassigned literal

Unit Propagation Rule

$$[I] \parallel F, C \vee (\neg)P \hookrightarrow [I, P(\text{or } \overline{P})] \parallel F, C \vee (\neg)P \text{ if } \begin{cases} [I] \not\models C \\ P \text{ undefined in } I \end{cases}$$

This is a restricted form of resolution

Example Revisited

$$F := \begin{array}{lll} C_1 = \neg P_1 \vee P_2 & C_2 = \neg P_3 \vee P_4 & C_3 = \neg P_6 \vee \neg P_5 \vee \neg P_2 \\ C_4 = \neg P_5 \vee P_6 & C_5 = P_5 \vee P_7 & C_6 = \neg P_1 \vee P_5 \vee \neg P_7 \end{array}$$

I	Rule
P_1°	Decide
P_1°, P_2	Propagate
$P_1^\circ, P_2, P_3^\circ$	Decide
$P_1^\circ, P_2, P_3^\circ, P_4$	Propagate
$P_1^\circ, P_2, P_3^\circ, P_4, P_5^\circ$	Decide
$P_1^\circ, P_2, P_3^\circ, P_4, P_5^\circ, \overline{P_6}$	Propagate
$P_1^\circ, P_2, P_3^\circ, P_4, \overline{P_5}$	Backtrack
$P_1^\circ, P_2, P_3^\circ, P_4, \overline{P_5}, P_7$	Propagate

I	Rule
$P_1^\circ, P_2, \overline{P_3}$	Backtrack
$P_1^\circ, P_2, \overline{P_3}, P_5^\circ$	Decide
$P_1^\circ, P_2, \overline{P_3}, P_5^\circ, \overline{P_6}$	Propagate
$P_1^\circ, P_2, \overline{P_3}, \overline{P_5}$	Backtrack
$P_1^\circ, P_2, \overline{P_3}, \overline{P_5}, P_7$	Propagate
$\overline{P_1}$	Backtrack
$\ldots$	
$\overline{P_1}, P_2^\circ, P_3^\circ, P_4, \overline{P_5}, P_7$	Sat

Example

$$F := \begin{array}{lll} C_1 = \neg P_1 \vee P_2 & C_2 = \neg P_2 \vee P_3 & C_3 = \neg P_3 \vee P_4 \\ C_4 = \neg P_4 \vee P_5 & C_5 = \neg P_5 \vee \neg P_1 & C_6 = P_1 \vee P_2 \vee P_3 \vee P_4 \vee \neg P_5 \end{array}$$

I	Rule
P_1°	Decide
P_1°, P_2	Propagate
P_1°, P_2, P_3	Propagate
P_1°, P_2, P_3, P_4	Propagate
$P_1^\circ, P_2, P_3, P_4, P_5$	Propagate
$\overline{P_1}$	Backtrack
$\overline{P_1}, P_2^\circ$	Decide
$\overline{P_1}, P_2^\circ, P_3$	Propagate
$\cdots$	(Several propagations)
$\overline{P_1}, P_2^\circ, P_3, P_4, P_5$	Sat

Non-Chronological Backtracking & Clause Learning

The backtracking rule seems short-sighted

- It always jumps to the most recent decision
- It does not keep information about the conflict

Backjump Rule

$$[I_1, P^\circ, I_2] \parallel F \hookrightarrow [I_1, \ell] \parallel F, C \text{ if } \begin{cases} [I_1, P^\circ, I_2] \not\models F \\ \text{Exists } C \text{ s.t. :} \\ \quad F \Rightarrow (C \rightarrow \ell) \\ \quad I_1 \models C \\ \quad \text{var}(\ell) \text{ undef. in } I_1 \\ \quad \text{var}(\ell) \text{ appears in } F \end{cases}$$

C is called a *conflict clause*

Will help us prevent similar conflicts in the future

Example Revisited (again)

$$F := \begin{array}{lll} C_1 = \neg P_1 \lor P_2 & C_2 = \neg P_3 \lor P_4 & C_3 = \neg P_6 \lor \neg P_5 \lor \neg P_2 \\ C_4 = \neg P_5 \lor P_6 & C_5 = P_5 \lor P_7 & C_6 = \neg P_1 \lor P_5 \lor \neg P_7 \end{array}$$

$$C_7 = \neg P_1 \lor \neg P_5$$

I	Rule
P_1°	Decide
P_1°, P_2	Propagate
$P_1^\circ, P_2, P_3^\circ$	Decide
$P_1^\circ, P_2, P_3^\circ, P_4$	Propagate
$P_1^\circ, P_2, P_3^\circ, P_4, P_5^\circ$	Decide
$P_1^\circ, P_2, P_3^\circ, P_4, P_5^\circ, \overline{P_6}$	Propagate
$P_1^\circ, P_2, \overline{P_5}$	Backjump, $P_1 \rightarrow \neg P_5$
$P_1^\circ, P_2, \overline{P_5}, P_7$	Propagate
$\overline{P_1}$	Backjump, *true* $\rightarrow \neg P_1$
$\ldots$	

Finding a Conflict Clause

The Backjump rule requires a conflict clause

To find one, we construct an *implication graph* $G = (V, E)$

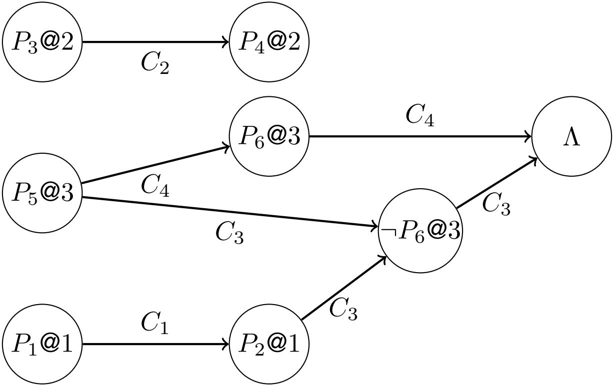
- V has a node for each decision literal in I, labeled with the literal's value and its decision level.
- For each clause $C = \ell_1 \vee \cdots \vee \ell_n \vee \ell$ where $\ell_1, \ldots, \ell_n$ are assigned *false*,
  1. Add a node for ℓ with the decision level in which it entered I
  2. Add edges (ℓ_i, ℓ) for $1 \leq i \leq n$ to E
- Add a special *conflict node* Λ. For any *conflict variable* with nodes labeled P and $\neg P$, add edges from these nodes to Λ in E.
- Label each edge with the clause that caused the implication.

The implication graph contains sufficient information to generate a conflict clause

Implication Graph

$$F := \begin{array}{lll} C_1 = \neg P_1 \lor P_2 & C_2 = \neg P_3 \lor P_4 & C_3 = \neg P_6 \lor \neg P_5 \lor \neg P_2 \\ C_4 = \neg P_5 \lor P_6 & C_5 = P_5 \lor P_7 & C_6 = \neg P_1 \lor P_5 \lor \neg P_7 \end{array}$$

$$I = [P_1^\circ, P_2, P_3^\circ, P_4, P_5^\circ, \overline{P_6}]$$



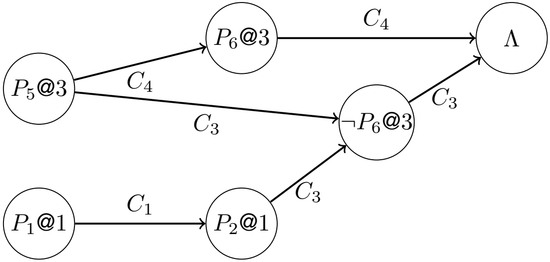
Conflict Graph

Implication graph where:

- Exactly one conflict variable
- All nodes have a path to Λ

$$C_1 = \neg P_1 \vee P_2 \quad C_2 = \neg P_3 \vee P_4$$
$$C_3 = \neg P_6 \vee \neg P_5 \vee \neg P_2$$
$$C_4 = \neg P_5 \vee P_6 \quad C_5 = P_5 \vee P_7$$
$$C_6 = \neg P_1 \vee P_5 \vee \neg P_7$$

$$I = [P_1^\circ, P_2, P_3^\circ, P_4, P_5^\circ, \overline{P_6}]$$



Generating Conflict Clauses

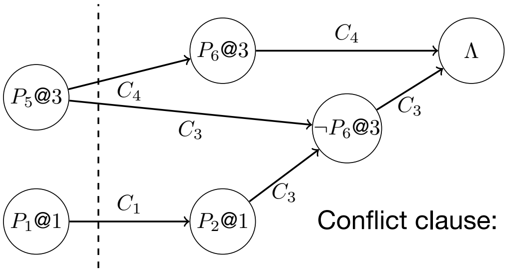
Consider a conflict graph G

1. Pick a cut in G such that:
   - All of the decision nodes are on one side (the “reason” side)
   - At least one conflict literal is on the other (the “conflict” side)
2. Pick all nodes K on the reason side with an edge crossing the cut
3. The nodes in K form a *cause* of the conflict
4. The negations of the corresponding literal form the conflict clause

Generating Conflict Clauses

$$C_1 = \neg P_1 \vee P_2 \quad C_2 = \neg P_3 \vee P_4$$
$$C_3 = \neg P_6 \vee \neg P_5 \vee \neg P_2$$
$$C_4 = \neg P_5 \vee P_6 \quad C_5 = P_5 \vee P_7$$
$$C_6 = \neg P_1 \vee P_5 \vee \neg P_7$$

$$I = [P_1^\circ, P_2, P_3^\circ, P_4, P_5^\circ, \overline{P_6}]$$



Conflict clause: $\neg P_1 \vee \neg P_5$

Generating Conflict Clauses

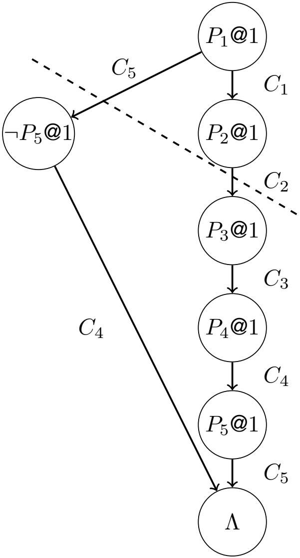
$$\begin{array}{ll} C_1 = \neg P_1 \vee P_2 & C_2 = \neg P_2 \vee P_3 \\ C_3 = \neg P_3 \vee P_4 & C_4 = \neg P_4 \vee P_5 \\ C_5 = \neg P_5 \vee \neg P_1 & \\ C_6 = P_1 \vee P_2 \vee P_3 \vee P_4 \vee \neg P_5 & \end{array}$$

$$I = [P_1^{\circ}, P_2, P_3, P_4, P_5]$$

Conflict clause: $P_1 \rightarrow \neg P_2$

Any others?

Does order matter?



Generating Conflict Clauses

This corresponds to resolution:

1. Let C be the conflicted clause
2. Pick most recently implied literal in conflict graph G
3. Let C' be the clause that implied it
4. Let $C \leftarrow \text{resolve}(C, C')$
5. Repeat step 2 while applicable

$$C_1 = \neg P_1 \vee P_2 \quad C_2 = \neg P_3 \vee P_4$$
$$C_3 = \neg P_6 \vee \neg P_5 \vee \neg P_2$$
$$C_4 = \neg P_5 \vee P_6 \quad C_5 = P_5 \vee P_7$$
$$C_6 = \neg P_1 \vee P_5 \vee \neg P_7$$

$$I = [P_1^\circ, P_2, P_3^\circ, P_4, P_5^\circ, \overline{P_6}]$$

1. $C = \neg P_5 \vee P_6$
2. Pick $\overline{P_6}$
3. $C' = \neg P_6 \vee \neg P_5 \vee \neg P_2$
4. $C = \neg P_5 \vee \neg P_2$
5. Pick P_2
6. $C' = \neg P_1 \vee P_2$
7. $C = \neg P_1 \vee \neg P_5$

Generating Conflict Clauses

The textbook doesn't cover this at all

For more information, see:

- `http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume22/beame04a-html/`, Sections 3.4 and 3.5
- *Decision Procedures* by Kroening and Strichman. Download a copy from the library by visiting: `http://vufind.library.cmu.edu/vufind/Record/1607216`

DPLL and CDCL

Original DPLL used:

Decide, Sat/Unsat, Propagate, Backtrack

Modern DPLL replaces:

Backtrack with Backjump

These are called *Conflict Driven Clause Learning* (CDCL) solvers

In addition, most use:

- "Forgetting": periodically forget learned clauses
- Restart: reset interpretation, but keep learned clauses

```
while(1) {
  while(exists_unit(I, F))
    I, F = propagate(I, F);
  I, F = decide(I, F);
  if(conflict(I, F)) {
    if(has_decision(I))
      I, F = backjump(I, F);
    else
      return unsat;
  } else if(sat(I, F))
    return sat;
}
```

Correctness of DPLL

Soundness

For every execution starting with $[\varnothing] \parallel F$ and ending with $[I] \parallel$ *sat* (resp. $[I] \parallel$ *unsat*), F is satisfiable (resp. unsatisfiable).

Completeness

If F is satisfiable (resp. unsatisfiable), then every execution starting with $[\varnothing] \parallel F$ ends with $[I] \parallel$ *sat* (resp. $[I] \parallel$ *unsat*).

Note: Termination not obvious with Backjump. Define a metric that decreases:

- When adding a decision level (Decide)
- When adding literal to the current decision level (Propagate)
- When adding literal to *previous* decision level (Backjump)

Practical Considerations

Conflict-Driven Clause Learning (CDCL) made large-scale SAT practical

- GRASP solver, 1996
- From hundreds and low-thousands to thousands and millions of variables
- Focus shifted towards better heuristics, implementation

Several considerations proved effective:

- Make resolution more efficient: keep # memory accesses per iteration low
- Simple, low-overhead decision guidance
- Strategies for forgetting learned clauses

Watch Pointers

Idea: Watch two unassigned literals in each non-satisfied clause. Ignore the rest.

Maintain two lists for each variable P

- The first, L_P, contains watching clauses with P
- The second, $L_{\overline{P}}$, contains watching clauses with $\neg P$

Each time an assignment to is made to P:

1. For clauses in $L_{\overline{P},P}$, find another literal in the clause to watch
2. If (1) is not possible, the clause is unit

Advantages:

1. When P assigned, only examine clauses in the appropriate list
2. No overhead when backtracking

Dynamic Largest Individual Sum (DLIS)

Decision heuristic: choose variable that satisfies the most clauses

How do we implement this?

- Maintain *sat* counters for every variable
- When clauses are satisfied, update counters
- Must touch every clause containing literal set to 1
- Need to reverse process when backtracking

More overhead than unit propagation...

Probably not worth it

Variable State Independent Decaying Sum (VSIDS)

Rank variables by literal count in the initial database

- Only increment when clauses are learned
- Periodically divide all counts by 2

Main idea: bias towards literals from recent conflicts

- Conflict adds 1 to each literal in conflict clause
- More time passed $\rightarrow$ more divisions by 2
- Effectively solves conflicts before moving onto new clauses

Use heap structure to find unassigned variable with the highest ranking

Other Approaches

There are other good SAT-solving approaches

Randomized approaches (GSAT, WSAT)

- Hill-climbing, local search algorithms
- State: full interpretation, Cost: # non-satisfied clauses
- Move: flip one assignment

Binary decision diagrams

- Efficiently represent formula as a DAG
- Manipulate formula by changing graph structure

Stalmarck's algorithm

- Breadth-first search: try both branches at once
- Also branch on variable relationships

Next Lecture

Install Dafny on your machine

See the **Assignments** section on course webpage for a guide